Lecture 3(Part1)

Topics covered:
CPU Architecture

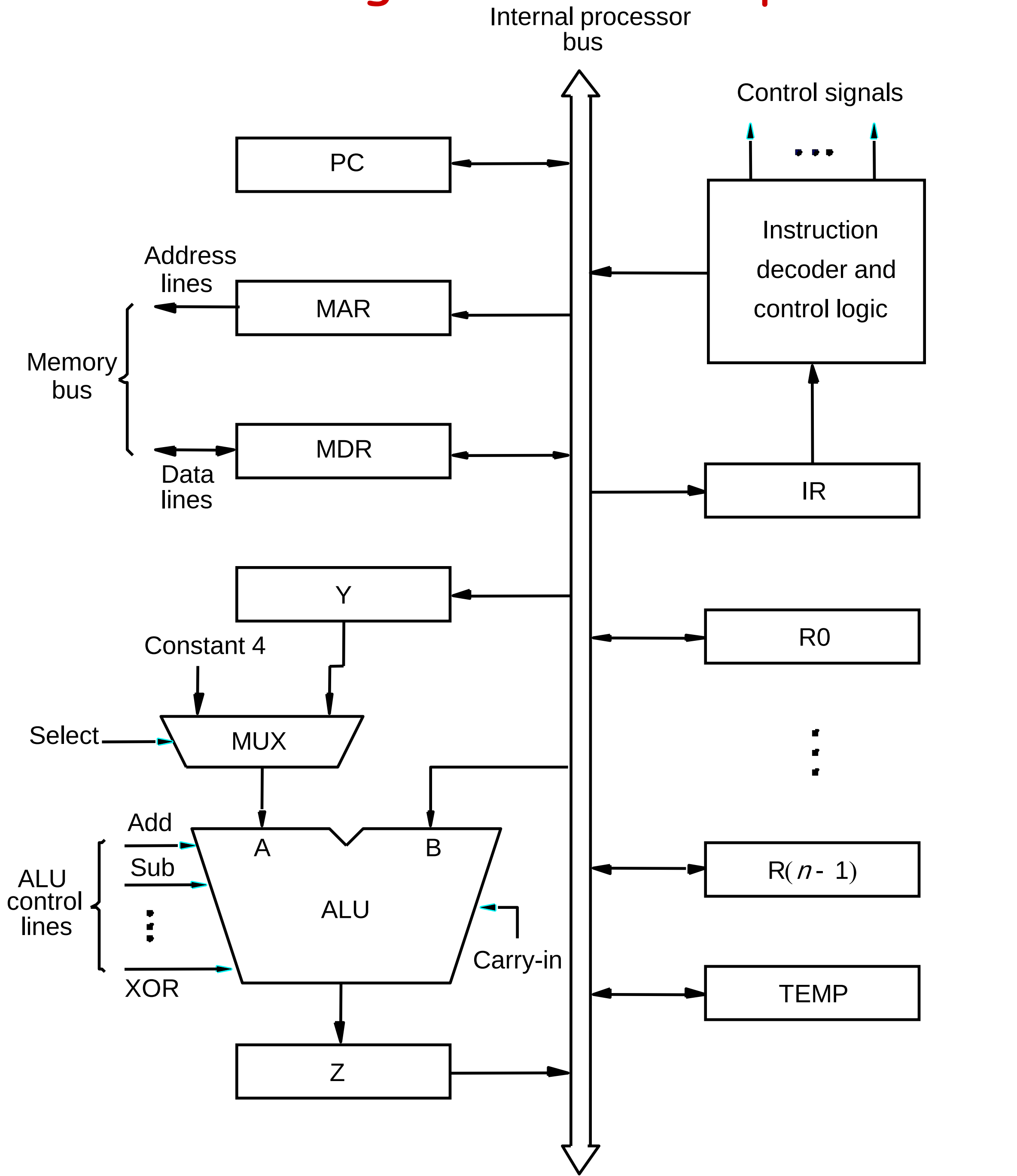
Fetch/execute cycle of an instruction

- Step I:
  - Fetch the contents of the memory location pointed to by Program Counter (PC).
  - PC points to the memory location which has the instruction to be executed.
  - Load the contents of the memory location into Instruction Register (IR).
- Step II:
  - Increment the contents of the PC by 4 (assuming the memory is byte addressable and the word length is 32 bits).
- Step III:
  - Carry out the operation specified by the instructions in the IR.
- Steps I and II constitute the fetch phase, and are repeated as many times as necessary to fetch the complete instruction.
- Step III constitutes the execution phase.

Internal organization of a processor

- Recall that a processor has several registers/building blocks:
  - Memory address register (MAR)
  - Memory data register (MDR)
  - Program Counter (PC)
  - Instruction Register (IR)
  - General purpose registers R0 - R(n-1)
  - Arithmetic and logic unit (ALU)
  - Control unit.
- How are these units organized and how do they communicate with each other?

Internal organization of a processor



Single bus organization

- Single bus organization:
  - ALU, control unit and all the registers are connected via a single common bus.
  - Bus is internal to the processor and should not be confused with the external bus that connects the processor to the memory and I/O devices.
- Data lines of the external memory bus are connected to the internal processor bus via MDR.
  - Register MDR has two inputs and two outputs.
  - Data may be loaded to (from) MDR from (to) internal processor bus or external memory bus.
- Address lines of the external memory bus are connected to the internal processor bus via MAR.
  - MAR receives input from the internal processor bus.
  - MAR provides output to external memory bus.

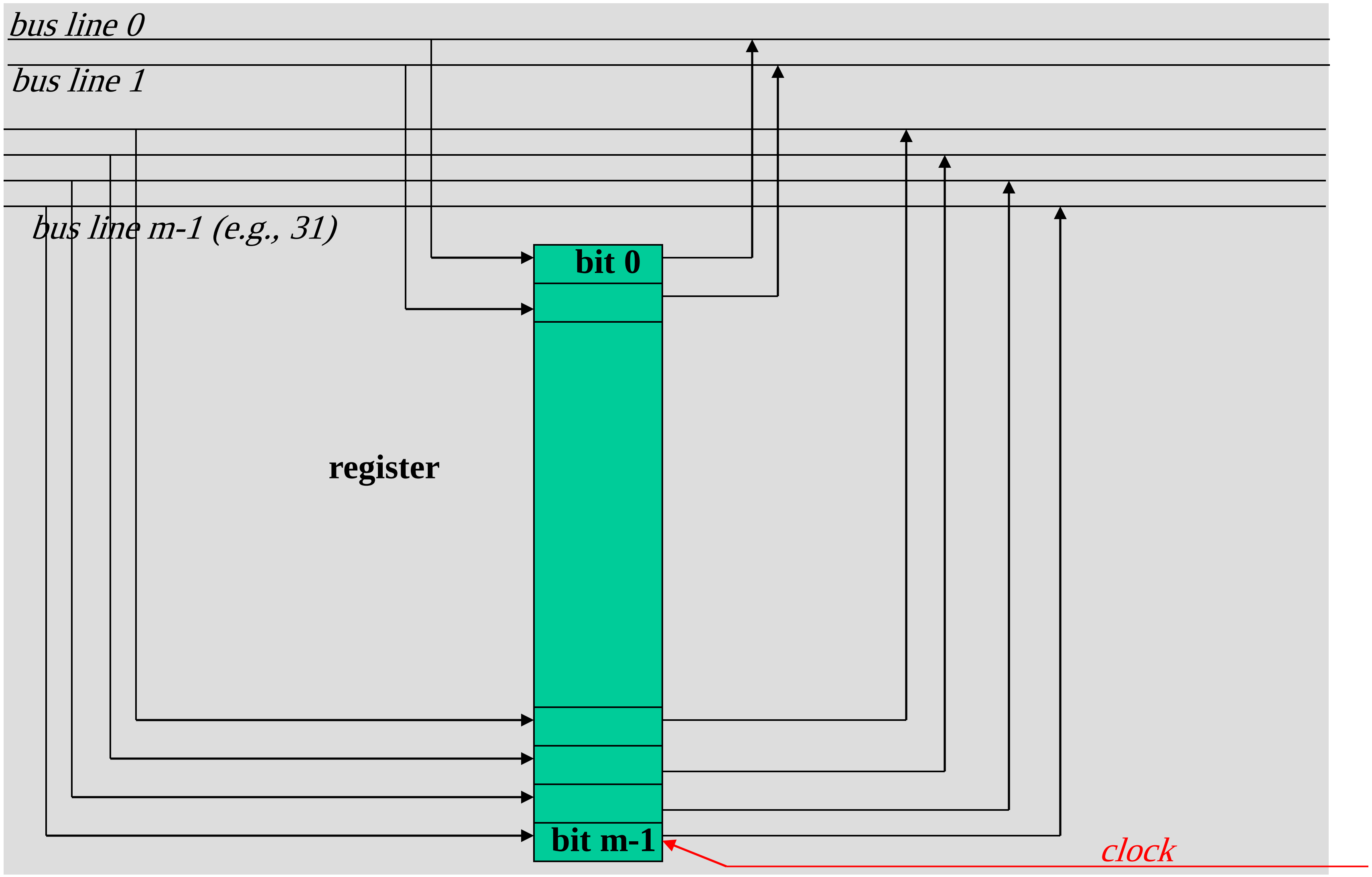
Single bus organization (contd..)

- Instruction decoder and control logic block, or control unit issues signals to control the operation of all units inside the processor and for interacting with the memory bus.
  - Control signals depend on the instruction loaded in the Instruction Register (IR)
- Outputs from the control logic block are connected to:
  - Control lines of the memory bus.
  - ALU, to determine which operation is to be performed.
  - Select input of the multiplexer MUX to select between Register Y and constant 4.
  - Control lines of the registers, to select the registers.

Single bus organization (contd..)

- Registers Y, Z, and TEMP:
  - Used by the processor for temporary storage <u>during</u> execution of some instructions.
  - Note that Registers R0 to R(n-1) are used to store data generated by one instruction for later use by another instruction.
  - Data is stored in R0 through R(n-1) <u>after</u> the execution of an instruction.
- Multiplexer MUX selects either the output of register Y or a constant 4, depending upon the control input Select.
  - Constant 4 is used to increment the value of the PC.

Registers and the bus

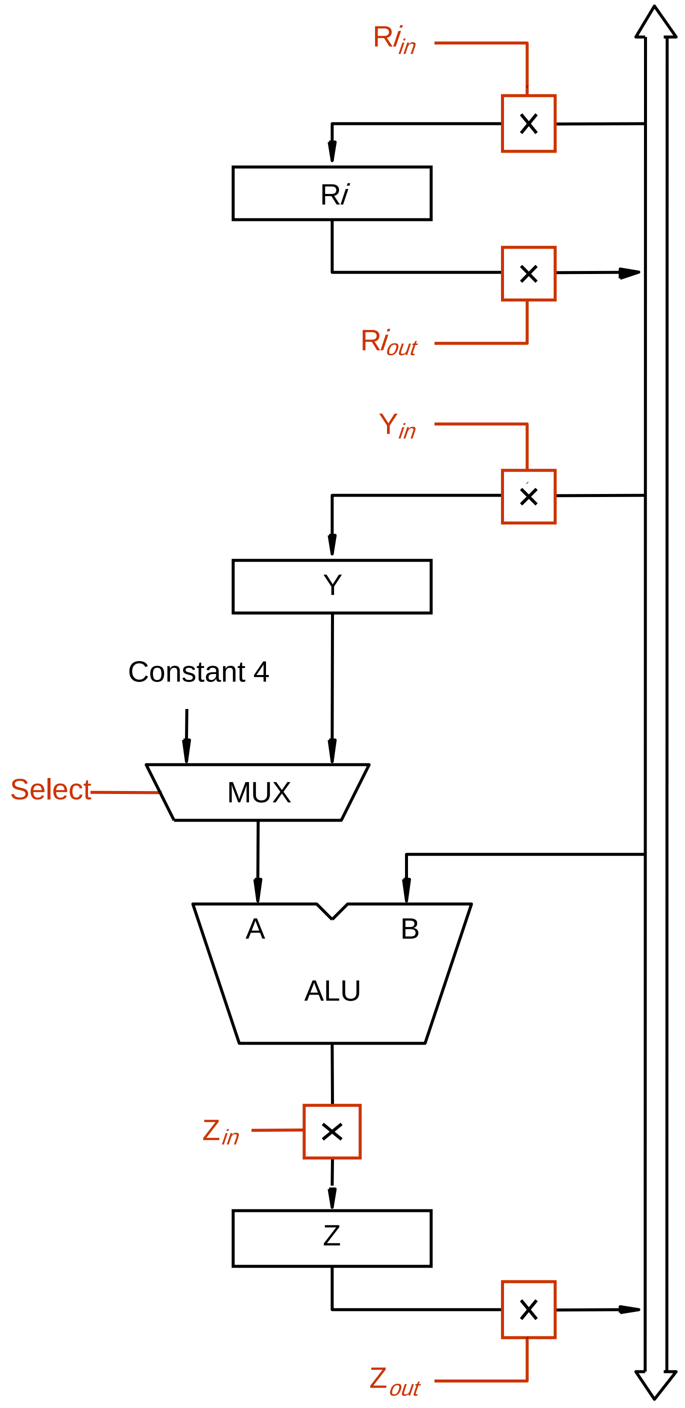


Registers and the bus (contd..)

- A bus may be viewed as a collection of parallel wires.
- Buses have no memory:
  - They are just a collection of wires.
- When data is on the bus, all registers can "see" that data at their inputs.
- A register may place its contents onto the bus.

Registers and the bus (contd..)

- At any one time, only one register may output its contents to the bus:
  - Which register outputs its content to the bus is determined by the control signal issued by the control logic.
  - Control signal depends on the instruction loaded in the instruction register IR.
- Registers can load data from the bus:
  - Which registers load data from the bus is determined by the control signal issued by the control logic.
- Registers are clocked (sequential) entities (unlike ALU which is purely combinatorial).



Registers are connected to the bus via switches controlled by the signals Rin & Rout.

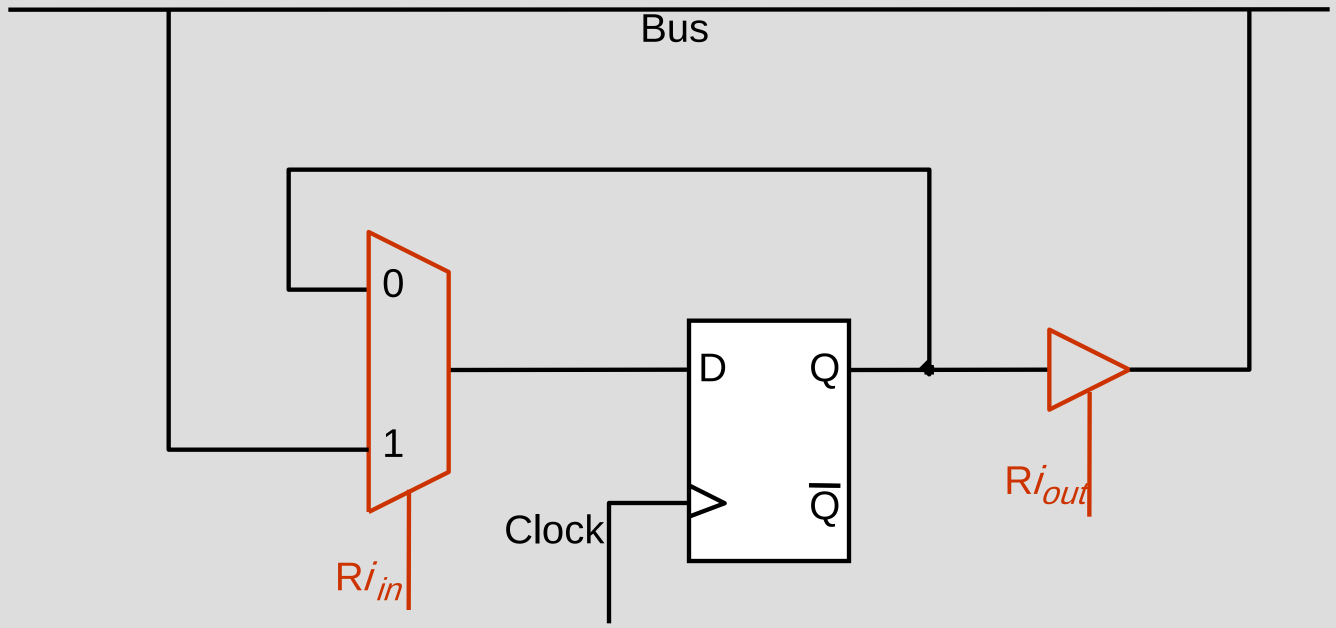
Each register *Ri* has two control signals, Ri_{in} and Ri_{out}.

If $Ri_{in}=1$, the data from the bus is loaded into the register.

If $Ri_{out}=1$, the data from the register is loaded onto the bus.

The same holds for registers *Y* and *Z* as well.

Registers and the bus (contd..)

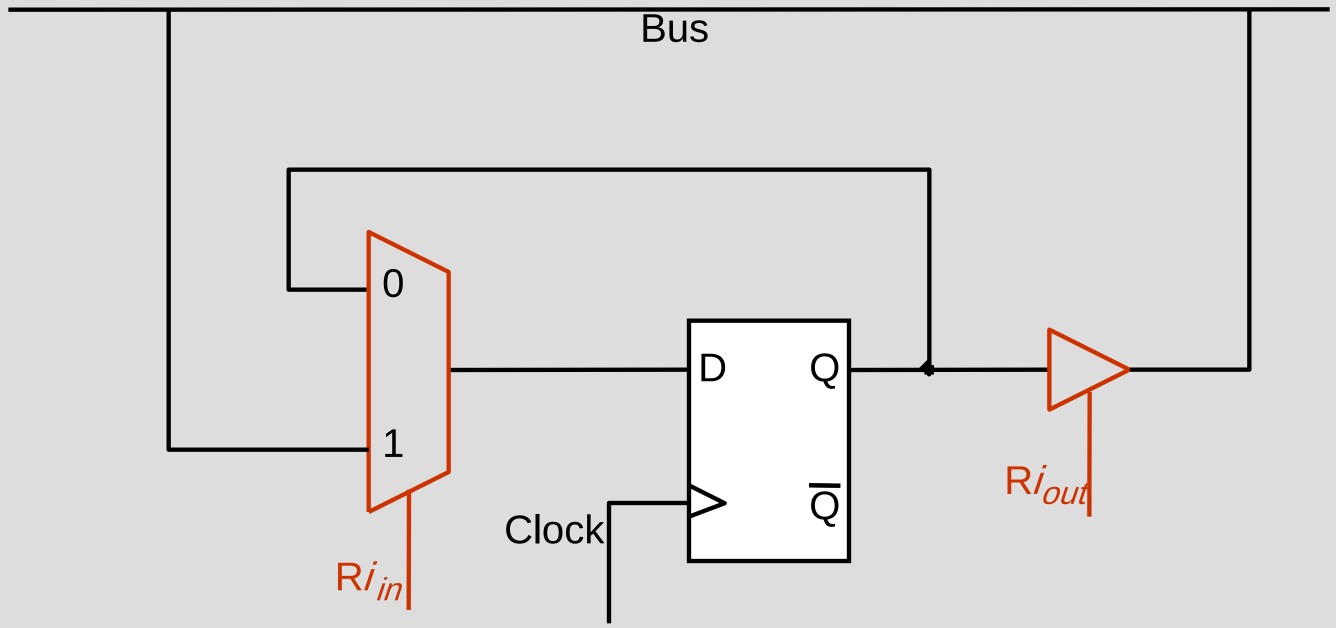


• *Each bit in a register may be implemented by an edge-triggered D flip flop.*

• *Two input multiplexer is used to select the data applied to the input of an edge triggered flip-flop.*

• *Q output of the flip-flop is connected to the bus via a tri-state gate.*

Registers and the bus (contd..)



$Ri_{in} = 1$:

Multiplexer selects the data on the bus.

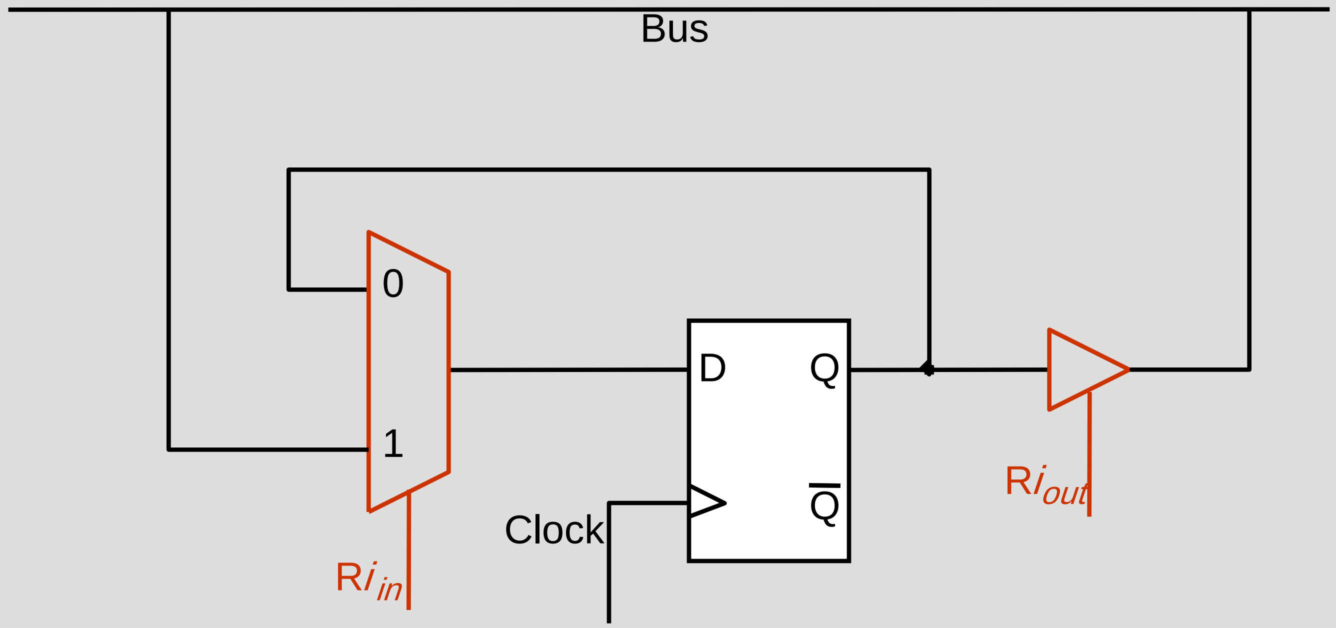
Data is loaded into the flip-flop at the rising edge of the clock.

$Ri_{in} = 0$:

Multiplexer feeds back the value currently stored in the flip-flop.

Q output represents the value currently stored in the flip-flop.

Registers and the bus (contd..)



Ri_{out} = 1:

Tri-state gate loads the value of the flip-flop onto the bus.

Data is loaded onto the bus at the rising edge of the clock.

Ri_{out} = 0:

Gate's output is in high-impedance (electrically disconnected) state.

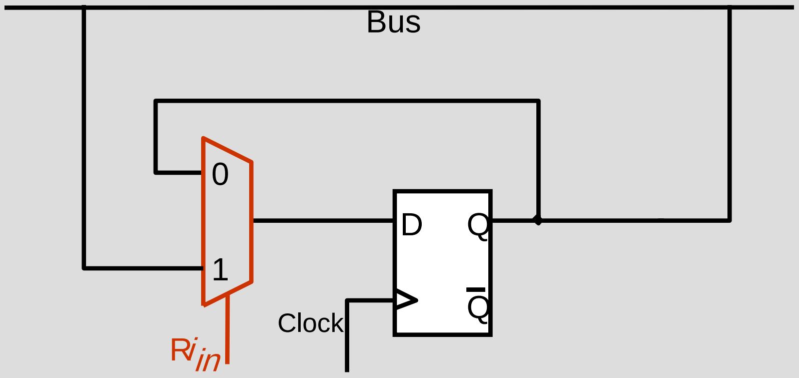
Corresponds to open-circuit state.

Registers and the bus (contd..)

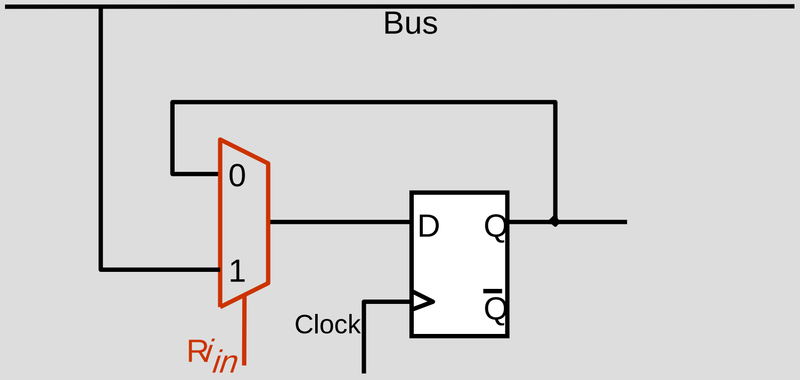
Operation of a tri-state gate

- *A tri-state gate can enter one of three output states.*
  - *its output can be in a logic low state (L).*
  - *its output can be in a logic high state (H).*
  - *its output can be effectively an open-circuit (high impedance)*
- *When a tri-state gate is connected to a bus in high-impedance state, its outputs are effectively disconnected from the bus.*

Ri_{out} = 1, output is:
Logic low, if Q = 0
Logic high, if Q = 1

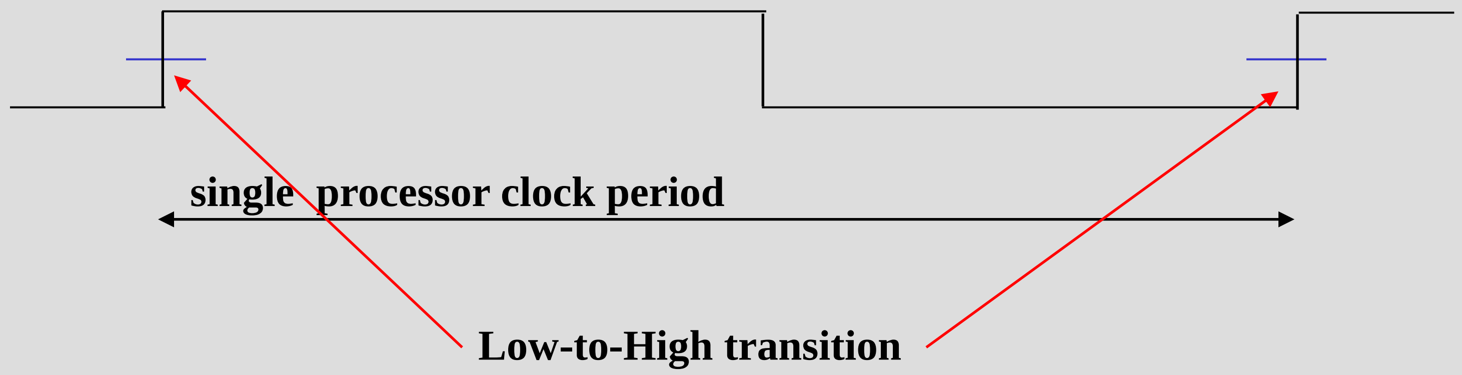


Ri_{out} = 0:
High impedance
Open circuit condition



Registers and the bus (contd..)

Operation of an edge-triggered flip-flop



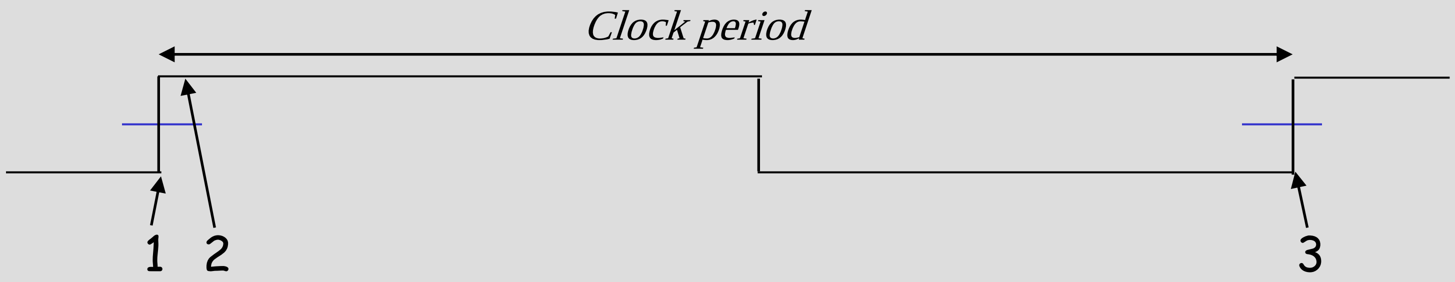
•*Data is loaded from the register to the bus (or to the register from the bus) at the rising edge of the clock.*
•*Data is loaded at the L-H transition of the clock.*

Registers and the bus (contd..)

- Data transfers and operations take place within time periods defined by the processor clock.
  - Time period is known as the clock cycle.
- At the beginning of the clock cycle, the control signals that govern a particular transfer are asserted.
  - For e.g., if the data are to be transferred from register R0 to the bus, then $R0_{out}$ is set to 1.
- Edge-triggered flip-flop operation explained earlier used only the rising edge of the clock for data transfer.
  - Other schemes are possible, for example, data transfers may use rising and falling edges of the clock.
- When edge-triggered flip-flops are not used, two or more clock signals may be needed to guarantee proper transfer of data. This is known as multiphase clocking.

Simple register transfer example

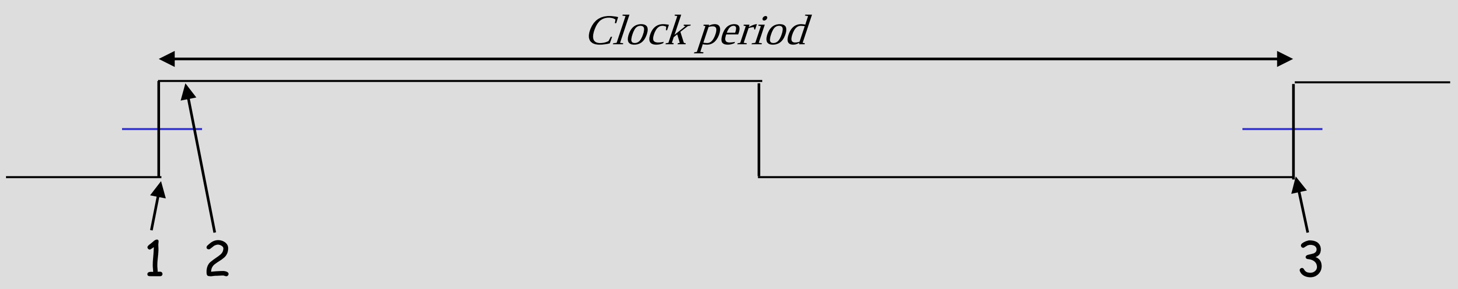
Transfer the contents of register R3 to register R4



1. Control signals $R3_{out}$ and $R4_{in}$ become 1. They stay valid until the end of the clock cycle.
2.After a small delay, the contents of R3 are placed onto the bus. The contents of R3 stay onto the bus until the end of the clock cycle.
3. At the end of the clock cycle, the data onto the bus is loaded into R4. $R3_{out}$ and $R4_{in}$ become 0.

Loading multiple registers from the bus

Transfer the contents of register R3 to register R4, R5



1. Control signals $R3_{out}$, $R4_{in}$ and $R5_{in}$ become 1. They stay valid until the end of the clock cycle.
2.After a small delay, the contents of R3 are placed onto the bus. The contents of R3 stay onto the bus until the end of the clock cycle.
3. At the end of the clock cycle, the data onto the bus is loaded into R4 and R5. $R3_{out}$, $R4_{in}$ and $R5_{in}$ become 0.

Loading multiple registers from the bus (contd..)

- It is possible to load multiple registers simultaneously from the bus.
  - For e.g., transfer the contents of register R3 to registers R4 and R7 simultaneously.
- The number of registers that can be simultaneously loaded depends on:
  - Drive capability (fan-out)
  - Noise.
  - Note that this is an electrical issue, not a logical issue.
- Distinguish this from multiple registers loading the bus:
  - For e.g. load the contents of registers R3 and R4 onto the bus simultaneously.
  - Logically inconsistent event.
  - Physically dangerous event.

Arithmetic Logic Unit (ALU)

- ALU is a purely combinatorial device:
  - It has no memory or internal storage.
- It has 2 input vectors:
  - These may be called the A- and B-vector or the R- and S-vector
  - The inputs are as wide as the registers/system bus (e.g., 16, 32 bits)
- It has 1 output vector
  - Usually denoted F

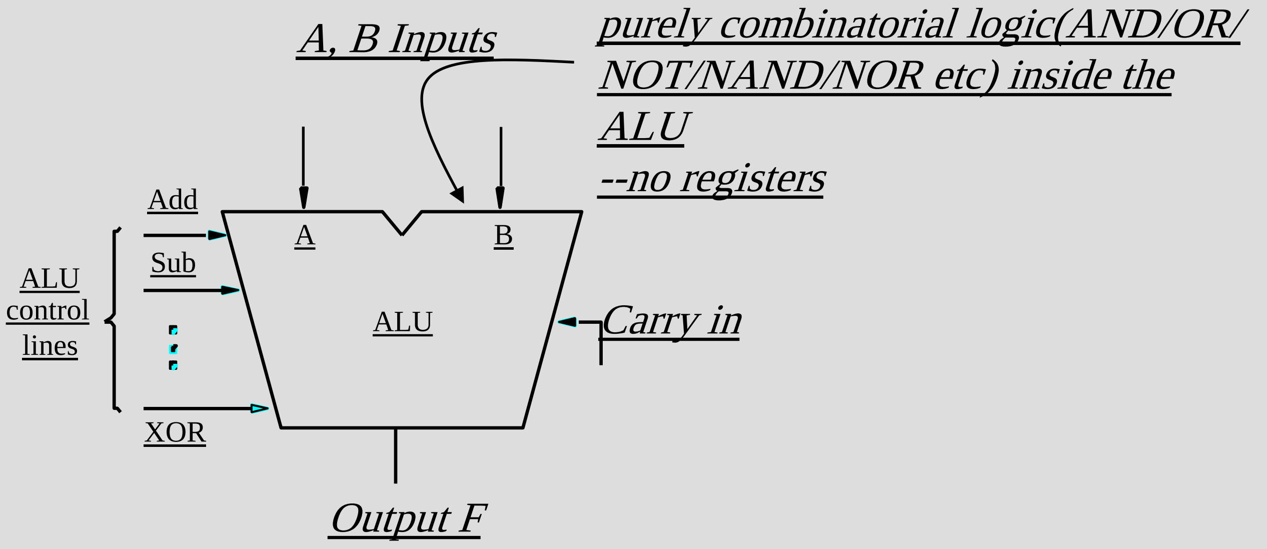
Arithmetic Logic Unit (ALU) (contd..)

Sample functions performed by the ALU

- F = A+B F = A+B+1
- F = A-B F = A-B-1
- F = A and B F = A or B
- F = not A F = not B
- F = not A + 1 F = not B + 1
- F = (not A) and BF = A and (not B)
- F = A xor B F = not (A xor B)
- F = A F = B

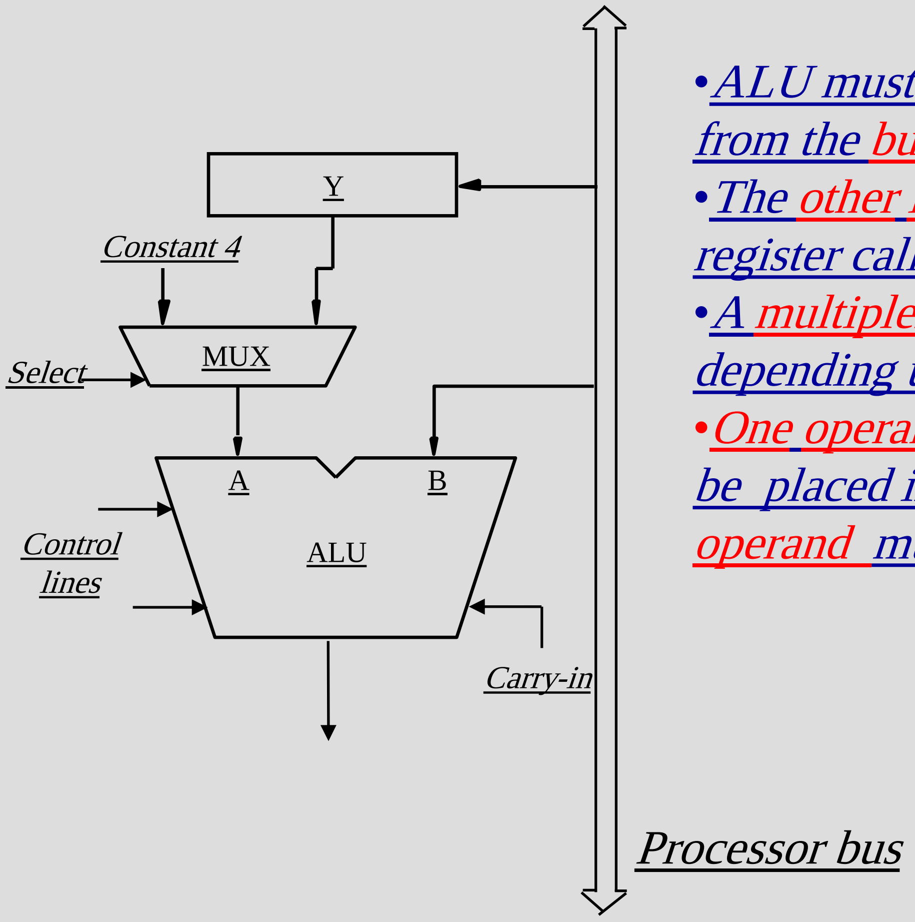
Arithmetic Logic Unit (ALU) (contd..)

ALU is basically a black-box



Arithmetic and Logic Unit (ALU) (contd..)

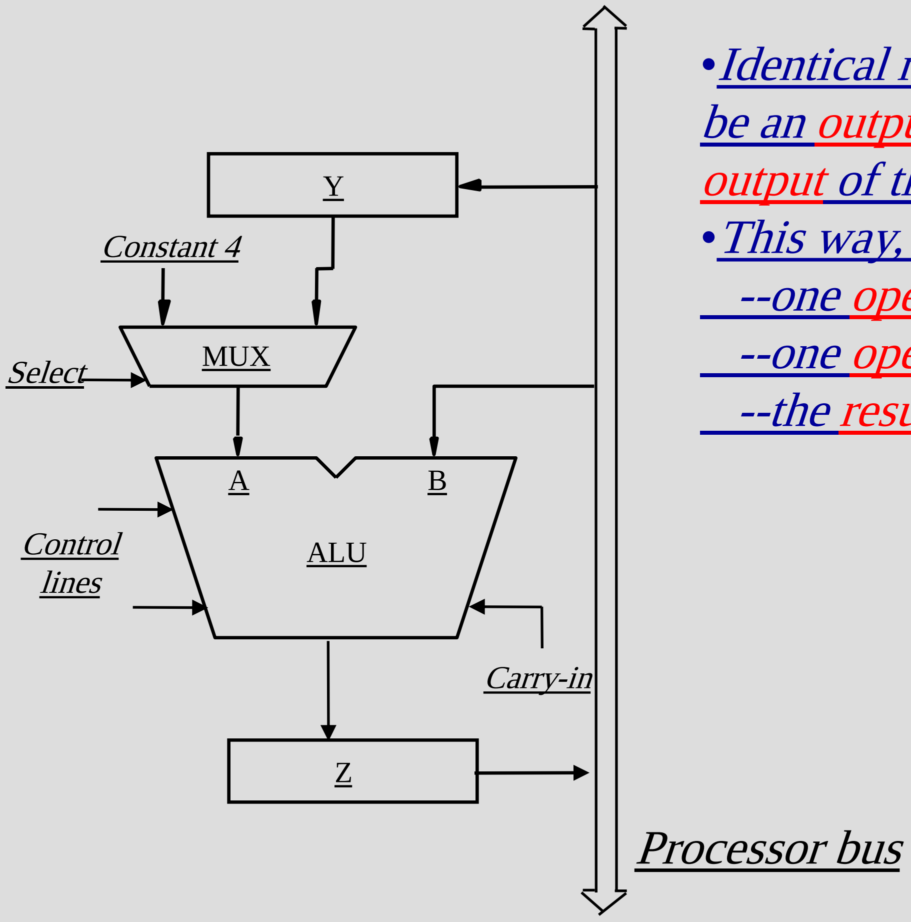
ALU connections to the bus



- *ALU must have only one input connection from the bus.*
- *The other input must be stored in a holding register called Y register.*
- *A multiplexer selects among register Y and 4 depending upon select line.*
- *One operand of a two-operand instruction must be placed into the Y register before the other operand must be placed onto the bus.*

Arithmetic and Logic Unit (ALU) (contd..)

ALU connections to the bus



- *Identical reasoning tells us that there must be an output register Z which collects the output of the ALU at the end of each cycle.*
- *This way, there can be*

  --one operand in the Y register

  --one operand ON THE BUS

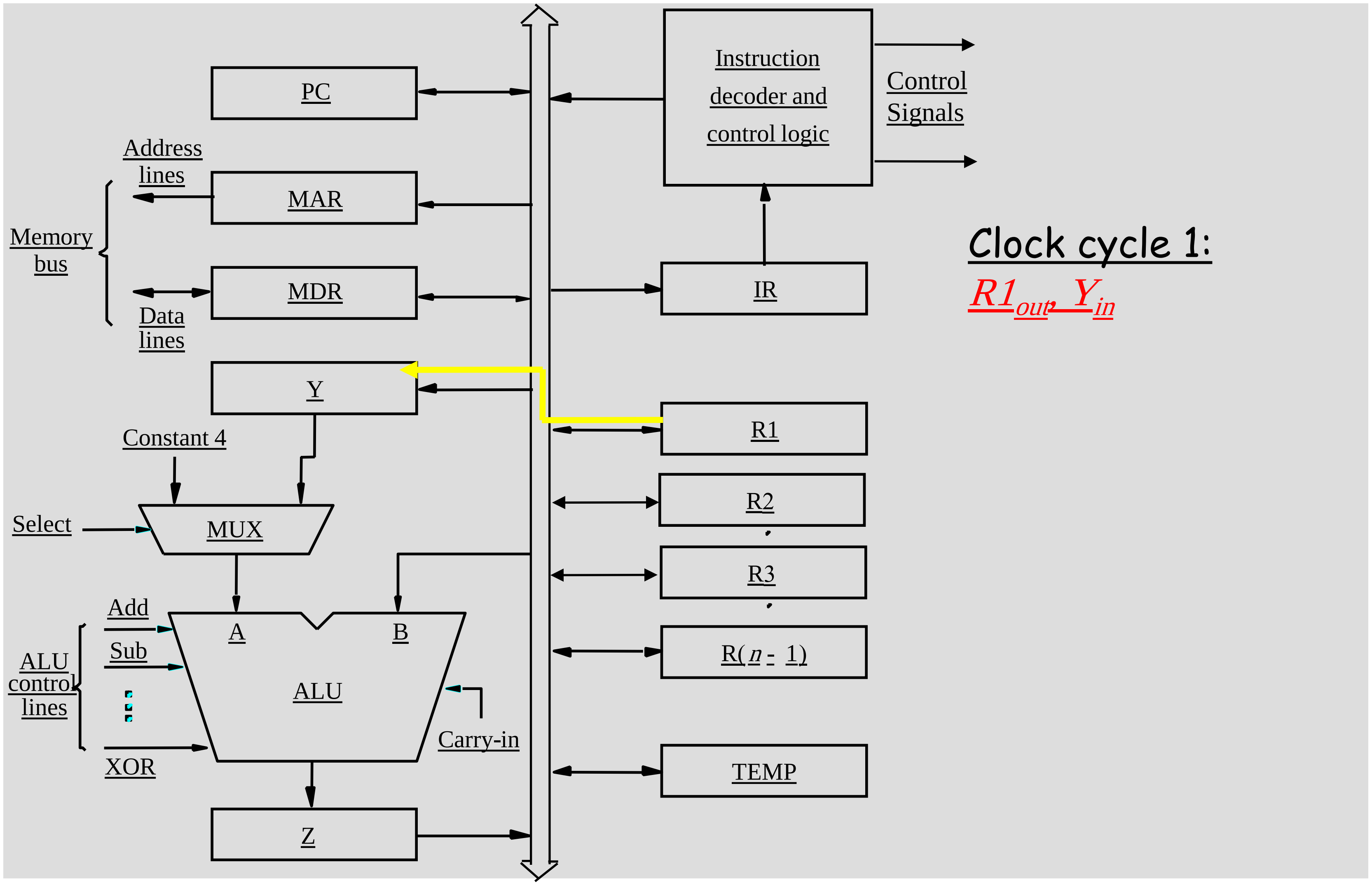
  --the result stored in the Z register

Performing an arithmetic operation

Add the contents of registers *R1* and *R2* and place the result in *R3*.
That is: *R3 = R1 + R2*

1. Place the contents of register R1 into the Y register in the first clock cycle.
2. Place the contents of register R2 onto the bus in the second clock cycle.
Both inputs to the ALU are now valid. Select register Y, and assert the ALU command F=A+B.
3. In the third clock cycle, Z register has latched the output of the ALU. Thus the contents of the Z register can be copied into register R3.

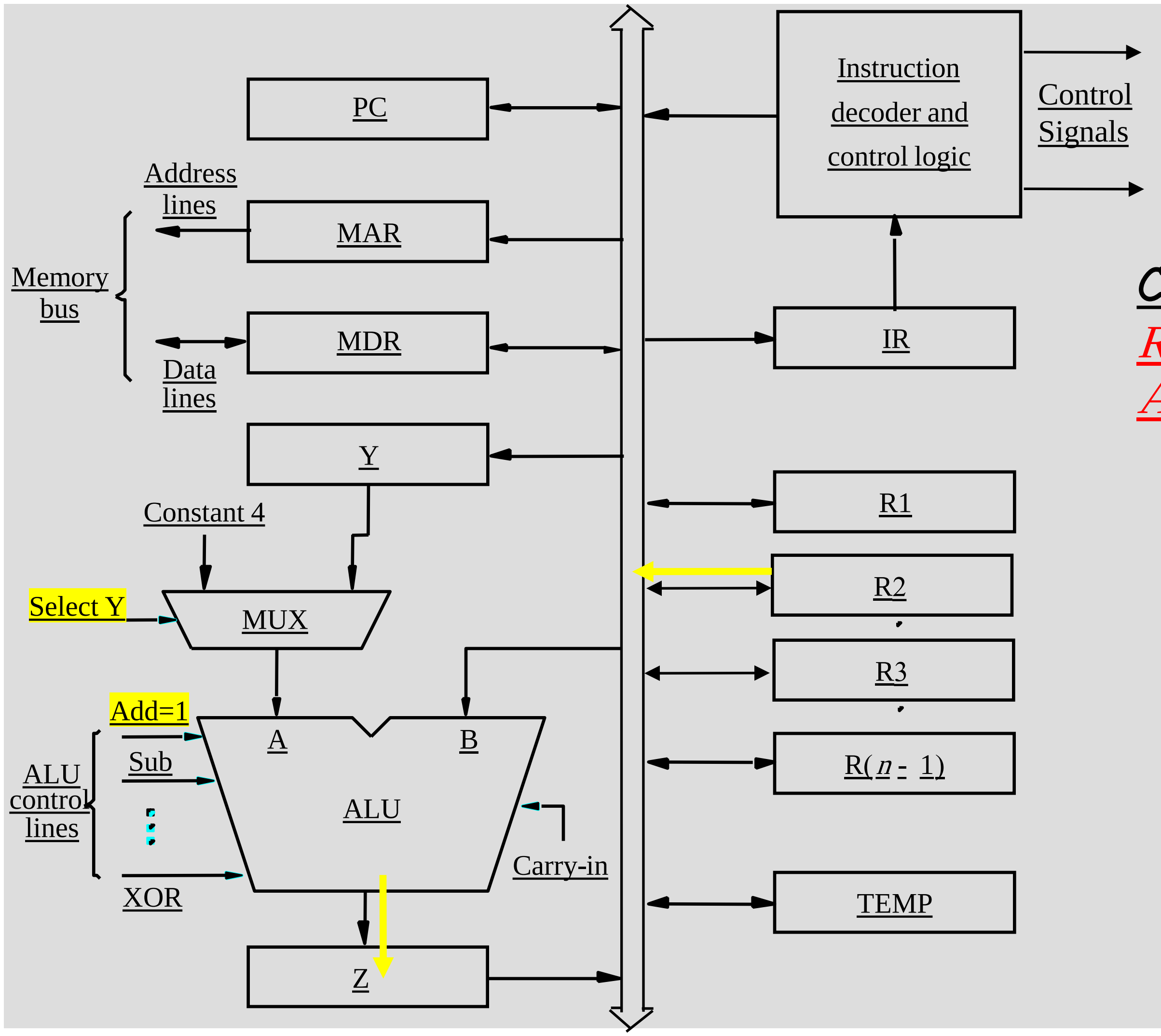
Performing an arithmetic operation (contd..)



Clock cycle 1:

$R1_{out}$, Y_{in}

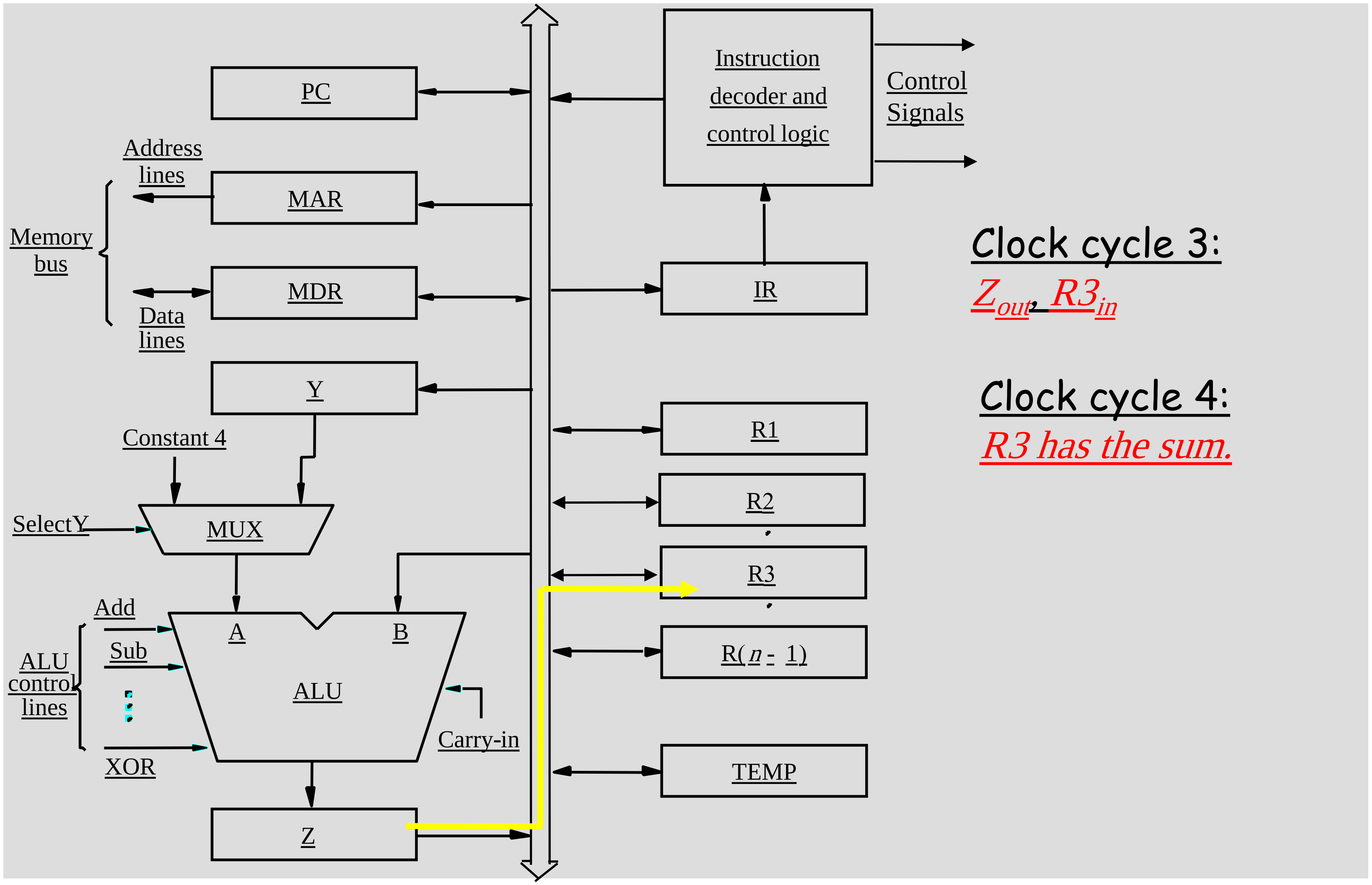
Performing an arithmetic operation (contd..)



Clock cycle 2:

$R2_{out}$, SelectY, Add, Z_{in}

Performing an arithmetic operation (contd..)



Clock cycle 3:

Z_{out}, $R3_{in}$

Clock cycle 4:

R3 has the sum.

Performing an arithmetic operation (contd..)

Clock Cycle 1:

$R1_{out}$, Y_{in} (Y=R1)

Clock Cycle 2:

$R2_{out}$, SelectY, Add, Z_{in} (Z = R1+R2)

Clock Cycle 3:

Z_{out}, $R3_{in}$ (R3=Z)

Performing an arithmetic operation (contd..)

- Inputs of the ALU:
  - Input B is tied to the bus.
  - Input A is tied to the output of the multiplexer.
- Output of the ALU:
  - Tied to the input of the Z register.
- Z register:
  - Input tied to the output of the ALU.
  - Output tied to the bus.
  - Unlike Ri_{in}, Z_{in} loads data from the output of the ALU and not the bus.

Performing an arithmetic operation (contd..)

Events as seen by the system registers, bus, ALU over time.

cycle:	controls active	what the bus sees	what Y has	output of ALU
start of 1	$R1_{out}$, Y_{in}	contents of R1	--unknown	--unknown
end of 1 = start of 2	$R1_{out}$, Y_{in}	contents of R1	R1	--unknown
start of 2	$R2_{out}$, F=A+B	contents of R2	R1	F=A+B=R1+R2 but this is not valid yet
end of 2 = start of 3	$R2_{out}$, F=A+B,Z_{in}	contents of R2	R1	F=A+B=R1+R2 (now valid)
start of 3	Z_{out},$R3_{in}$	contents of Z	R1	--unknown
end of 3	Z_{out}, $R3_{in}$	contents of Z	R1	--unknown (but R3 latches bus contents)

ALU operations

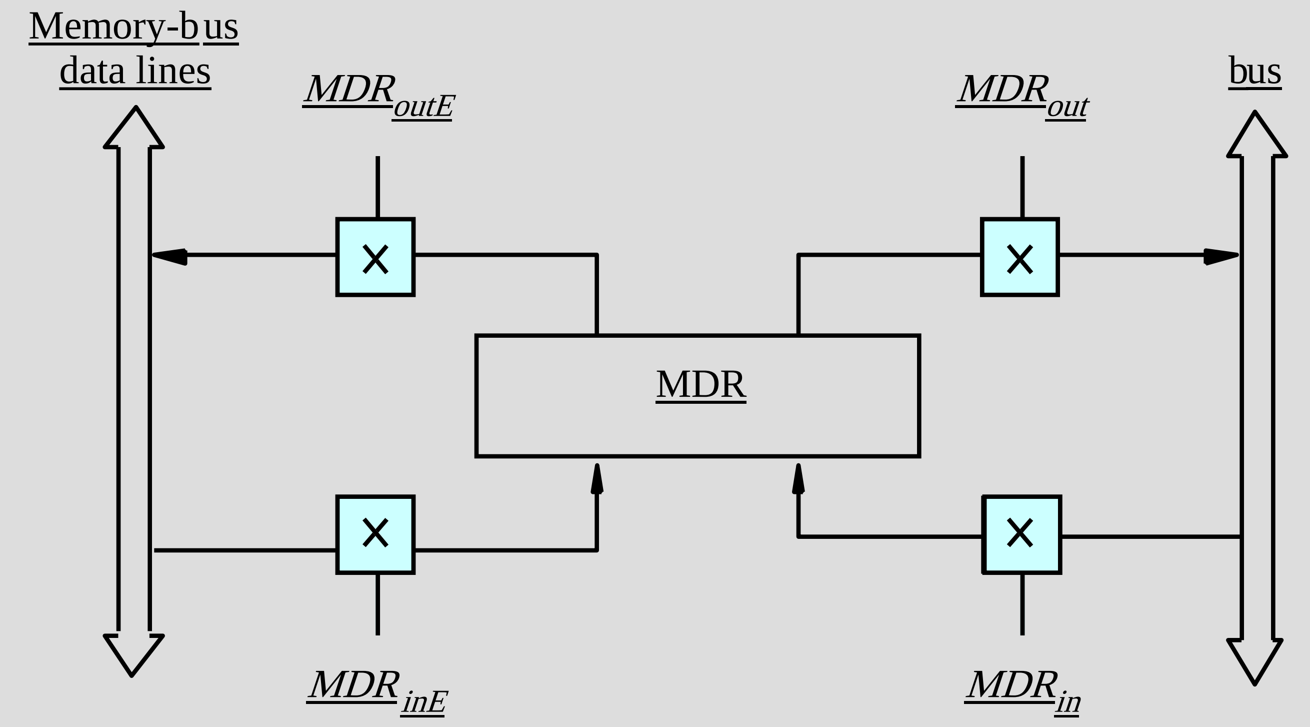
- $RC = RA\ op\ RB$
- Clock cycle 1:
  - Move RA to Y register.
  - RA_{out}, Y_{in}
- Clock cycle 2:
  - Put RB on the bus, perform $F = RA\ op\ RB$, and transfer the result to Z.
  - $RB_{out}, (RA\ op\ RB)=1, Select Y, Z_{in}$
- Clock cycle 3:
  - Put Z on the bus, and load Z into RC.
  - Z_{out}, RC_{in}

Fetching a word from memory

- Processor has to specify the address of the memory location where this information is stored and request a Read operation.
- Processor transfers the required address to *MAR*.
  - Output of *MAR* is connected to the address lines of the memory bus.
- Processor uses the control lines of the memory bus to indicate that a *Read* operation is needed.
- Requested information are received from the memory and are stored in *MDR*.
  - Transferred from *MDR* to other registers.

Fetching a word from memory (contd..)

Connections for register MDR



MDR_{outE} and MDR_{inE} control connection to external bus.

MDR_{out} and MDR_{in} control connection to internal bus.

Fetching a word from memory (contd..)

- Timing of the internal processor operations must be coordinated with the response time of memory Read operations.
- Processor completes one internal data transfer in one clock cycle.
- Memory response time for a Read operation is variable and usually longer than one clock cycle.
  - Processor waits until it receives an indication that the requested Read has been completed.
  - Control signal Memory Function Completed (MFC) is used for this purpose.
  - **MFC** is set to **1** by the memory to indicate that the contents of the specified location have been read and are **available** on the data lines of the memory **bus**.

Fetching a word from memory (contd..)

MOVE (R1), R2

1. Load the contents of Register *R1* into *MAR*.
2. Start a *Read* operation on the memory bus.
3. Wait for *MFC* response from the memory.
4. Load *MDR* from the memory bus.
5. Load the contents of *MDR* into Register *R2*.

Steps can be performed separately, some may be combined.

1. Steps 1 and 2 can be combined.
   - Load ***R1*** to ***MAR*** and activate ***Read*** control signal simultaneously.
2. Steps 3 and 4 can be combined.
   - Activate control signal ***MDR_{inE}*** while waiting for response from the memory MFC.
3. Last step loads the contents of ***MDR*** into Register ***R2***.

Hence, Memory ***Read*** operation takes 3 steps.

Fetching a word from memory (contd..)

MOVE (R1) , R2: Memory operation takes 3 steps.

Step 1:

- Place *R1* onto the internal processor bus.
- Load the contents of the bus into *MAR.*
- Activate the *Read* control signal.
- ***$R1_{out}$, MAR_{in}, Read.***

Step 2:

- Wait for MFC from the memory.
- Activate the control signal to load data from external bus to *MDR.*
- ***MDR_{inE}, WMFC***

Step 3:

- Place the contents of *MDR* onto the internal processor bus.
- Load the contents of the bus into Register *R2.*
- ***MDR_{outP}, $R2_{in}$***

Storing a word into memory

MOVE R2, (R1): Memory operation takes 3 steps.

Step 1:

- Place *R1* onto the internal processor bus.
- Load the contents of the internal processor bus into *MAR.*
  - ***$R1_{out}$, MAR_{in}.***

Step 2:

- Place *R2* onto the internal processor bus.
- Load the contents of the internal processor bus into *MDR*.
- Activate Write operation.
  - ***$R2_{out}$, MDR_{in}, Write***

Step 3:

- Place the contents of *MDR* into the external memory bus.
- Wait for the memory write operation to be completed MFC.
  - ***MDR_{outE}, WMFC***

Execution of a complete instruction

Add the contents of a memory location pointed to by Register *R3* to register *R1.*

ADD (R3), R1

To execute the instruction we must execute the following tasks:

1. Fetch the instruction.
2. Fetch the operand (contents of the memory location pointed to by *R3*.)
3. Perform the addition.
4. Load the result into *R1*.

Execution of a complete instruction

Task 1: Fetch the instruction

Recall that:

- *PC* holds the address of the memory location which has the next instruction to be executed.
- *IR* holds the instruction currently being executed.

Step 1

- Load the contents of *PC* to *MAR*.
- Activate the *Read* control signal.
- Increment the contents of the *PC* by 4.
  - PC_{out}, MAR_{in}, ***Read, Select4, Add,*** Z_{in}.

Step 2

- Update the contents of the *PC*.
- Copy the updated *PC* to Register *Y* (useful for Branch instructions).
- Activate the control signal to load data from external bus to *MDR*
- Wait for *MFC* from memory.
  - Z_{out}, PC_{in}, Y_{in}, MDR_{inE}, ***WMFC***

Step 3

- Place the contents of *MDR* onto the bus.
- Load the *IR* with the contents of the bus.
  - MDR_{out}, IR_{in}

Execution of a complete instruction (contd..)

Task 2. Fetch the operand (contents of memory pointed to by *R3*.)
Task 3. Perform the addition.
Task 4. Load the result into *R1*.

Step 4: - Place the contents of Register *R3* onto internal processor bus.
- Load the contents of the bus onto *MAR*.
- Activate the *Read* control signal.
- ***$R3_{out}$, MAR_{in}, Read***

Step 5: - Place the contents of *R1* onto the bus.
- Load the contents of the bus into Register *Y* (Recall one operand in *Y*).
- Wait for *MFC*.
- ***$R1_{out}$, Y_{in}, MDR_{inE}, WMFC***

Step 6: - Load the contents of *MDR* onto the internal processor bus.
- Select *Y*, and perform the addition.
- Place the result in *Z*.
- ***MDR_{out}, SelectY, Add, Z_{in}.***

Step 7: - Place the contents of Register *Z* onto the internal processor bus.
- Place the contents of the bus into Register *R1*.
- ***Z_{out}, $R1_{in}$***

Execution of a complete instruction (contd..)

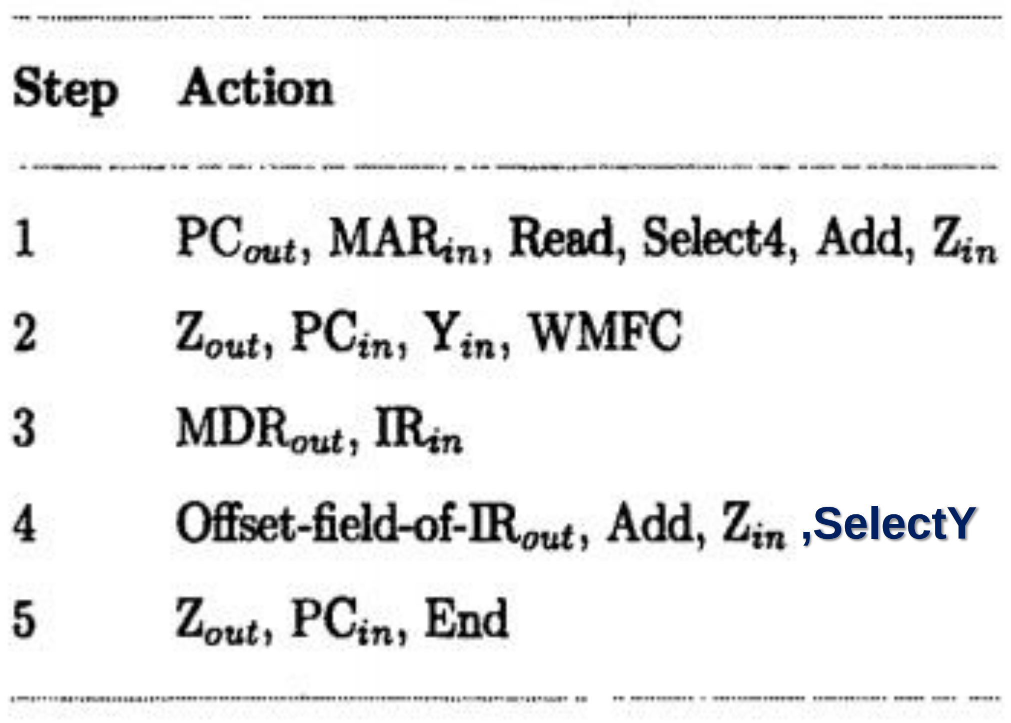
ADD (R3), R1

Step	Action
1	PC_{out} , MAR_{in} , Read, Select4, Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMF C
3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMF C
6	MDR_{out} , SelectY, Add, Z_{in}
7	Z_{out} , $R1_{in}$, End

Branch instructions

- Recall that the updated contents of the *PC* are copied into Register *Y* in Step 2.
  - Not necessary for *ADD* instruction, but useful in *BRANCH* instructions.:
  - Branch target address is computed by adding the updated contents of the *PC* to an offset.
- Copying the updated contents of the *PC* to Register *Y* speeds up the execution of *BRANCH* instruction.
- Since the Fetch cycle is the same for all instructions, this step is performed for all instructions.
  - Since Register *Y* is not used for any other purpose at that time it does not have any impact on the execution of the instruction.

Unconditional Branch instructions



Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Add, Z_{in} ,SelectY
5	Z_{out}, PC_{in}, End

Figure 7.7 Control sequence for an unconditional Branch instruction.

Conditional Branch instructions

Consider now a conditional branch. In this case, we need to check the status of the condition codes before loading a new value into the PC. For example, for a Branch-on-negative (Branch<0) instruction, step 4 in Figure 7.7 is replaced with

Offset-field-of-IR_{out}, Add, Z_{in} ,SelectY, If N = 0 then End

Thus, if N=0 the processor returns to step 1 immediately after step 4. If N=1, step 5 is performed to load a new value into the PC, thus performing the branch operation.